Demonstrating custom SIMD instruction development for a RISC-V softcore

Philippos Papaphilippou, Paul H. J. Kelly, Wayne Luk
Department of Computing, Imperial College London, UK
{pp616, p.kelly, w.luk}@imperial.ac.uk

***Abstract*—This demo elaborates on the programmability aspect of Simodense, a recently released open-source softcore, optimised for evaluating custom SIMD instructions. CPUs featuring small reconfigurable areas for implementing custom instructions is an alternative path in computer architecture that can help with the challenges found in today's FPGAs. By providing RTL-based programmability for implementing custom SIMD instructions, highly-integrated accelerators can be developed, while benefiting from the pre-existing CPU logic, such as the caches and their high memory throughput to main memory.**

***Index Terms*—softcore, custom SIMD instructions, template**

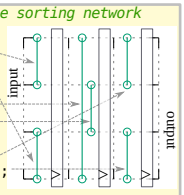
FPGAs are an exciting platform for a variety of applications. However, the current way of integrating them into systems has a performance overhead on main memory bandwidth and latency. This limitation is known to impact big data analytics accelerators. On the other hand, modern general purpose processors (CPUs) support a wide range of single-instruction-multiple data (SIMD) instructions as a way to accelerate applications that exhibit data-level parallelism. While this can be advantageous in certain applications, the instruction set extensions become bloated with overspecialised instructions [1], and sometimes it is difficult to express efficiently a parallel task using a fixed set of instructions.

As a solution to this problem, a possible future for microprocessor architecture is to include small FPGAs working as instructions. The literature on reconfigurable SIMD instructions is rather limited, with some early experimentation such as [2], which does not target high-performance and streaming applications. Moreover, since all communication is hardened and abstracted in CPUs, complex SIMD instructions can be implemented with only a few RTL lines of code, while achieving a higher operating frequency and performance than FPGA designs running as co-processors.

In combination with the openness of the RISC-V instruction set, and its support for custom instructions, now is an appropriate time to explore reconfigurable SIMD instructions, as demonstrated here with the use of the proposed open-source[1] RISC-V-based softcore [3].

This base example of this demo is a SIMD instruction implementation for sorting 4 integers 32-bit wide, using the provided template. Algorithm 1 shows how a few lines of Verilog can emulate a sorting network, such as the bitonic sorter. There are 3 sets of pipeline registers, and this latency is declared in line 3, as the template is applicable to instruction implementations of an arbitrary pipeline length. Lines 23 to 30 describe the 3 pipeline stages, which consist of compare-and-swap (CAS) units. The functionality of the CAS units is to sort 2 numbers (i.e. $a, b \rightarrow min(a,b), max(a,b)$) and the implementation is shown in lines 37 to 49. Since this instruction uses only one vector register for input and one for output, the non-applicable operands are greyed out (aliased with 0 in software). This instruction can then be conveniently called through inline assembly in software written in C/C++.

```
1  `define XLEN 32  // 32-bit base registers
2  `define VLEN 128 // 128-bit vector registers
3  `define c1_cycles 3 // pipeline length of c1 custom instruction
4
5  module c1 [...]
6      input clk, reset,  in_valid; // valid bit for input
7      input [4:0] rd;  input [2:0] vrd1, vrd2;
8                     // Destination register names (1 base and 2 vector)
9      input [`XLEN-1:0] in_data; // 32-bit input
10     input [`VLEN-1:0] in_vdata1, in_vdata2; // 128-bit input
11
12     output out_v; output [4:0] out_rd; output [2:0] out_vrd1, out_vrd2;
13                // (Delayed) output valid bit and output register names
14     output  [`XLEN-1:0] out_data; // 32-bit output
15     output [`VLEN-1:0] out_vdata1, out_vdata2; // 128-bit output
16
17     // Shift register logic to delay rd, vrd1, vrd2 and in_valid
18     // by c1_cycles, to out_rd, out_vrd1, out_vrd2 and out_v resp.
19     [...]
20     //// User code ////
21 wire [31:0] net [15:0]; // 32-bit wires inside the sorting network
22
23 CAS cas0(clk, net[0], net[1], net[4], net[5]);
24 CAS cas1(clk, net[2], net[3], net[6], net[7]);
25
26 CAS cas2(clk, net[4], net[7], net[8], net[11]);
27 CAS cas3(clk, net[5], net[6], net[9], net[10]);
28
29 CAS cas4(clk, net[8], net[9], net[12], net[13]);
30 CAS cas5(clk, net[10], net[11], net[14], net[15]);
31
32 // Assigning input and output to wires in the sorting network
33 // (only using 1 input and 1 output reg. for this instruction)
34 for (i=0; i<4; i=i+i) assign net[i]=in_vdata1[32*(i+1)-1-:32];
35 assign out_vdata1 = {net[12], net[13], net[14], net[15]};
36 endmodule // c1
```




```
37 // Compare-and-swap (CAS) unit implementation
38 module CAS (clk, inA, inB, outA, outB);
39     input clk;
40     input [31:0] inA, inB;
41     output reg [31:0] outA, outB;
42     always @(posedge clk) begin
43         if (inA<inB) begin
44             outA<=inA; outB<=inB;
45         end else begin
46             outB<=inA; outA<=inB;
47         end
48     end
49 endmodule // CAS
```

Algorithm 1: Verilog template, with user code in yellow

[1]**Source available:** https://github.com/pphilippos/simodense

References

[1] N. Dao, A. Attwood, B. Healy, and D. Koch, "Flexbex: A risc-v with a reconfigurable instruction extension," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 190–195.

[2] Ordaz, Jose Raul Garcia and Koch, Dirk, "A Soft Dual-Processor System with a Partially Run-Time Reconfigurable Shared 128-Bit SIMD Engine," in *29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–8.

[3] P. Papaphilippou, P. H. J. Kelly, and W. Luk, "Simodense: a RISC-V softcore optimised for exploring custom SIMD instructions," in *31st International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2021 (Accepted).